Definition and Terminology of Approximate Matching

Approximate matching is a technique for identifying similarities between or among digital objects, such as files, storage media, or network streams, which do not match exactly. The purpose of this special publication is to present a framework of definitions and terminology to facilitate communication between users and developers of the technology and build awareness in potential user communities.

This publication first presents definitions of the types of approximate matching; it goes on to describe the three approximate matching techniques that are typically recognized. The general properties of approximate matching algorithms are introduced, followed by a more specific definition in terms of their essential characteristics. Finally, the document presents a collection of use cases.

Introduction

The idea of approximate matching is conceptually related to that of exact matching using cryptographic hashes. Both techniques use a relatively short, fixed- or somewhat variable length bitstring as a proxy for a digital object of arbitrary size, such as a file, a storage medium or a network stream. These proxies then form the basis for comparison of the objects, with the degree of similarity of the proxies being an accurate predictor of the similarity of the original objects.

Match Types

The various types of approximate match are described below. An approximate matching algorithm should be able to perform at least one of the following matching functions:

- *Object similarity detection:* identify related objects, e.g. different versions of a text document or images that have been slightly modified to avoid detection by exact matching.
- *Embedded object detection:* identify a given object inside a container, e.g. a JPEG image within a word processor document or other file in a memory dump.
- *Fragment detection:* identify an original input based on a fragment, e.g. network packet analysis or identification of cropped images.

- *Cross correlation:* identify files that share a common object, e.g. a word processor document and a network trace file containing the same JPEG image.

Matching Techniques

In general, approximate matching techniques can be broken down into three categories that are distinguished by the features used to measure similarity between objects:

Semantic Matching

Semantic matching uses contextual attributes of the digital object, and operates at a level close to human perception. For example, facial recognition algorithms look for patterns in images that resemble human faces. The algorithm is not concerned with the format of the image files it uses or indeed the fact that they are files at all.

Perceptual hashing and *robust hashing* are well-known terms for semantic matching.

Syntactic Matching

Syntactic matching uses internal structures present in digital objects. For example, the structure of a TCP network packet is defined as an international standard and matching tools can make use of this structure during network packet analysis to match the source, destination or content of the packet. Similar techniques may be applied to the internal structures of files and even storage media.

Syntactic matching algorithms typically run some syntactic pre-processing and then apply bytewise algorithms to the results, for example by removing the header information from a network packet and applying a bytewise matching algorithm to the packet contents.

Bytewise Matching

Bytewise matching refers to the assessment of the similarity of data objects at the bit stream level. It does not rely on semantic comparison of documents, images and sounds, etc. Neither does it rely on syntactic comparison of the structure of objects, such as packet structure during network packet analysis. That it not to say that bytewise approximate matching cannot be used in these scenarios, only that bytewise approximate matching does not rely on syntactic or semantic features of the data object.

Common terms in the literature for bytewise approximate matching are *fuzzy hashing, similarity hashing* and *similarity digest*.

Matching Algorithms

While matching techniques as described above may differ, the structure and output of the approximate matching algorithms are the same. The output of an approximate matching algorithm is referred to as a *similarity digest.* This corresponds to the familiar cryptographic digests or hashes used in exact matching.

Like a cryptographic hash, a similarity digest is a compact representation of the original data object that is suitable for comparison with the similarity digests of other data objects. The need for compression arises from the need to be able quickly to compare essential qualities of various data objects in lieu of brute forcing comparisons on the entire data object.

Since the approximate matching algorithm needs to generate and compare similarity digests, it consists of two functions:

A **feature extraction function** extracts some features or attributes from each object and combines them to produce a compressed representation of the original object. How features are picked and interpreted depends on the algorithm. The collection of features is the similarity digest of the object; the number or fraction of features that are shared by objects defines those objects' similarity.

A **similarity function** is the function used to compare similarity digests. Common convention suggests that the function should output a score, $\boldsymbol{s}$, which is scaled such that $\mathbf{0} \leq \boldsymbol{s} \leq \mathbf{100}$ where 0 indicates no similarity and 100 indicates high similarity.

Essential Characteristics of Matching Algorithms

As with traditional hash functions, there are some characteristics that should be demonstrated by approximate matching algorithms. These characteristics are described below.

Each algorithm should define how it incorporates each item and how it satisfies the reporting requirements for those items that have them.

Compression
A compact similarity digest is desirable, as it will usually allow a faster comparison while requiring less storage space. Ideally the digest will have a fixed length like traditional cryptographic hash values. For a constant efficiency and reliability of the results (see below) the shorter a digest value, the better.

> **NOTE:** *The similarity digests produced by current bytewise approximate matching algorithms are not of a fixed length but of a proportional length*[1].

Ease of computation
An algorithm should include the results of testing speed based attributes. Speed based attributes include measures of runtime efficiency of the *feature extraction function* and runtime efficiency of the *similarity function*. The former might be expressed as a comparison with SHA-1 (there is often a trade-off between granularity and runtime efficiency of the processing).

[1]In fact the output is almost fixed length, but this comes with a security risk.

Similarity preservation
An algorithm should yield similar similarity digests for similar inputs and should define how it measures similarity. The similarity measure may include multiple attributes, and should be accompanied by a measure of the accuracy of the matching technique under the circumstances in which it is designed to be used, in addition to the nature of the matching score and a margin of error or confidence level. The algorithm should also state whether it identifies exact matches as such.

Reliability of results
The reliability of the results for a given technique depends on three factors. Each algorithm should define how it incorporates each item and how it satisfies the reporting requirements for those items that have them.

- **Sensitivity and robustness**
  An algorithm should provide some measure of its robustness. A technique's robustness will define the operating conditions in which it can function effectively, also called its performance envelope. For example, robustness addresses the minimum and maximum object sizes that an algorithm can reliably distinguish between.

- **Precision and recall**
  An algorithm should include a description of the method for determining reliability and test data. Specifically, it should include whether the test data is culled from existing collections or was developed to specifically support testing. The results of any tests may include precision & recall rates and false positive and false negative rates.

- **Security of results**
  An algorithm should indicate whether it includes security properties designed to prevent attacks. Such attacks include manipulation of the matching technique or input data such that a data object appears dissimilar to another object to which it is similar or similar to another object with which it has little in common.

Use Cases

The following use cases are intended to demonstrate the utility of approximate matching.

Enhanced Blacklisting. Objects that resemble known threats may themselves constitute threats. Thus a minor mutation in a piece of known malware may also be malware which will not be detected by exact matching against a blacklist, but may well be detected as an approximate match to the original.

Enhanced Whitelisting. Note that approximate matching is not well suited to traditional whitelisting. Similarity to a known good object does not guarantee acceptability of the object at hand. For example, a weaponized version of the *ssh* daemon may well show up as similar to the true daemon and be passed OK by whitelisting filter augmented with approximate matching.

However, if using a whitelist to root out weaponized versions of known software, approximate matching can be applied very successfully.

Illicit Image Detection. Given a set of target images, approximate matching can be used to search for manipulated variants of images in the set, fragments of known images, or images secreted in other files such as music files.

Intellectual Property Enforcement. As with illicit image detection, approximate matching can be used in the detection of manipulated or hidden copies of copyrighted works including documents, images, audio files and movies.

Object Grouping. Given a collection of objects, approximate matching can be used to sift out similar objects into groups. This is useful in the detection of, for example, email exchanges, particular in situations where the initial dataset is very large and disjoint, as in a cloud based email services or collections of network streams.